Helium API Documentation

0.0.2

BugFree Software

19/11/2013

The helium.api module contains the implementation and API of Helium. It is a simple Python API that makes specifying web automation cases as simple as describing them to someone looking over their shoulder at a screen.

The public functions and classes of Helium are listed below. If you wish to use Helium functions in your Python scripts you can import them from the `helium.api` module:

```
from helium.api import *
```

class `Alert` (*text=None*)
Bases: `helium.api.GUIElement` (page 3)

Lets you identify and interact with Javascript alert boxes.

`exists`()
Evaluates to true if this GUI element exists.

`text`
Text displayed in the alert box.

class `Button` (*text=None*, ***kwargs*)
Bases: `helium.api.HTMLElementContainingText`

Lets you identify a button by its name and read its properties. An example usage of 'Button' is:

```
Button("Log In").exists()
```

This will look for the HTML button with the 'Log In' label:

```
<button>Log in</button>
```

and will return True if found, False otherwise.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `HTMLElement` (page 3).

`center`
The center of this UI element, as a `helium.api.Point` (page 5).

`exists`()
Evaluates to true if this GUI element exists.

`height`
The height of this UI element.

`is_enabled`()
Returns true if this UI element can currently be interacted with.

`width`
The width of this UI element.

`x`
The x-coordinate of the top-left point of this UI element.

`y`
The y-coordinate of the top-left point of this UI element.

class `CheckBox` (*label=None*, ***kwargs*)
Bases: `helium.api.LabelledElement`

Lets you identify a check box (<input type="checkbox" /> HTML element) by its name or label and read its properties:

```
CheckBox("I agree").exists()
```

This looks for a check box with label 'I agree' and returns True if found, False otherwise.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `HTMLElement` (page 3).

center
The center of this UI element, as a helium.api.Point (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

is_checked()
Returns True if this GUI element is checked (selected).

is_enabled()
Returns True if this GUI element can currently be interacted with.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class ComboBox(*label=None*, ***kwargs*)
Bases: helium.api.LabelledElement

Lets you identify a combo box and read its properties and options. Eg.:

```
ComboBox("Language").exists()
```

This looks for a combo box with label 'Language' and returns True if found, False otherwise.

For an explanation of the parameters below, to_right_of, above and to_left_of, please see the documentation of HTMLElement (page 3).

center
The center of this UI element, as a helium.api.Point (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

is_readonly()
Returns True if the value of this GUI element cannot be modified.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class Config
Bases: object

This class contains Helium's run-time configuration. To modify Helium's behaviour, simply assign to the properties of this class. For instance:

```
Config.auto_wait_enabled = False
```

auto_wait_enabled = True
When auto-wait is enabled, Helium automatically waits after performing GUI actions, such as press() (page 9), click() (page 7) or write() (page 11). The amount of time waited is calculated dynamically. This can sometimes lead to wait intervals that are too long. To speed up

the execution of your scripts, the property `auto_wait_enabled` can therefore be used to disable Helium's automatic wait facility.

You can disable or enable Helium's auto-wait at any point in your script. For example:

```
>>> Config.auto_wait_enabled = False
>>> write("John", into="Name")
>>> press(TAB)
>>> write("Smith")
>>> Config.auto_wait_enabled = True
>>> click("Submit")
```

search_timeout_secs = 10
Amount of time (in seconds) Helium attempts to find a GUI element for. If the search does not succeed within the time defined by this configuration value, Automa raises `TimeoutExpired`. Increasing the property of this value can be useful on lower-spec computers.

class GUIElement
Bases: `object`

Base class for all Helium's GUI elements.

exists()
Evaluates to true if this GUI element exists.

class HTMLElement(*below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `helium.api.GUIElement` (page 3)

This class defines properties that are available for all of Helium's GUI (HTML) elements.

One feature that all of Helium's GUI elements support are the optional arguments `below=`, `to_right_of=`, `above=` and `to_left_of=`. These arguments specify where a particular GUI element is to be searched for. For example:

```
Button("OK", to_left_of="Cancel")
```

This identifies the Button to the left of text "Cancel". Being able to restrict the search region for a GUI element like this can be useful for disambiguating multiple occurrences of a GUI element, especially when the occurrences are arranged in a table.

Relative GUI element searches can be nested and combined arbitrarily with Helium's other functions. For example:

```
click(Button("Log in", to_right_of=TextField("Username:")))
```

This clicks on the button with text "Log in" to the right of text field "Username:".

center
The center of this UI element, as a `helium.api.Point` (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class Image(*alt*, ***kwargs*)
Bases: `helium.api.HTMLElementIdentifiedByXPath`

Lets you identifier an image (<img /> HTML element) by its alt parameter.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `HTMLElement` (page 3).

center
The center of this UI element, as a `helium.api.Point` (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class Label(*text=None*, ***kwargs*)
Bases: `helium.api.HTMLElementContainingText`

Lets you identify a label (<label></label> HTML element). For example:

```
Label("User name").exists()
```

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `HTMLElement` (page 3).

center
The center of this UI element, as a `helium.api.Point` (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class Link(*text=None*, ***kwargs*)
Bases: `helium.api.HTMLElementContainingText`

Lets you identify a link (<a> HTML element). For example:

```
Link("Next").exists()
```

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `HTMLElement` (page 3).

center
The center of this UI element, as a `helium.api.Point` (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class ListItem (*text=None*, ***kwargs*)
Bases: helium.api.HTMLElementContainingText

Lets you identify a list item by its name (HTML element).

For an explanation of the parameters below, to_right_of, above and to_left_of, please see the documentation of HTMLElement (page 3).

center
The center of this UI element, as a helium.api.Point (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class Point (*x=0*, *y=0*)
Bases: helium.api.Point (page 5)

A clickable point. To create a Point at an offset of an existing point, use + and −:

```
>>> point = Point(x=10, y=25)
>>> point + (10, 0)
Point(x=20, y=25)
>>> point - (0, 10)
Point(x=10, y=15)
```

count (*value*) → integer – return number of occurrences of value

index (*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

x
The x coordinate of the point.

y
The y coordinate of the point.

class RadioButton (*label=None*, ***kwargs*)
Bases: helium.api.LabelledElement

‘ Lets you identify a radio button (<input type=”radio” /> HTML element) by name or label and read its properties. To for instance check whether a radio button is currently selected, use:

```
RadioButton("Option 2").is_selected()
```

For an explanation of the parameters below, to_right_of, above and to_left_of, please see the documentation of HTMLElement (page 3).

center
The center of this UI element, as a helium.api.Point (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

is_selected()
Returns true if this radio button is selected.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class Text(*text=None*, ***kwargs*)
Bases: helium.api.HTMLElementContainingText

Lets you identify any text or label. This is most useful for checking whether a particular text is shown on the screen:

```
Text("Hello World!").exists()
```

For an explanation of the parameters below, to_right_of, above and to_left_of, please see the documentation of HTMLElement (page 3).

center
The center of this UI element, as a helium.api.Point (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

value
Returns the current value of this Text object.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class TextField(*label=None*, ***kwargs*)
Bases: helium.api.CompositeElement

Lets you identify a text field on the web page (<input /> or <textarea /> HTML elements). For example:

```
TextField("File name").value
```

This returns the value of the "File name" text field. If it is empty, the empty string '' is returned.

For an explanation of the parameters below, to_right_of, above and to_left_of, please see the documentation of HTMLElement (page 3).

center
The center of this UI element, as a helium.api.Point (page 5).

exists()
Evaluates to true if this GUI element exists.

height
The height of this UI element.

is_enabled()
Returns true if this UI element can currently be interacted with.

is_readonly()
Returns true if the value of this UI element cannot be modified.

value
Returns the current value of this text field. '' if there is no value.

width
The width of this UI element.

x
The x-coordinate of the top-left point of this UI element.

y
The y-coordinate of the top-left point of this UI element.

class Window(*title=None*)
Bases: helium.api.GUIElement (page 3)

Lets you identify a window by its title.

exists()
Evaluates to true if this GUI element exists.

handle
Returns the Windows Operating System handle (HWND) assigned to this window.

title
Returns the title of this Window.

attach_file(*file_path*, *to*)
Allows uploading a file, from local computer, to the <input type="file" /> element located on a website.

click(**elements*)

Parameters ***elements** – Comma-separated list of GUI elements, labels (strings) or points.

Clicks on the given elements in sequence. Common examples are:

```
click("Close")
click("File", "Save")
click("File", MenuItem("Save"))
click(Button("OK"))
click(Point(200, 300))
click(ComboBox("File type").center + (50, 0))
```

Strings passed as parameters are automatically wrapped as Text (page 6) objects (so click("Helium") is equivalent to click(Text("Helium"))).

doubleclick(*element*)

Parameters **element** – A GUI element, label (string) or point.

Performs a double-click on the given element. For example:

```
doubleclick("Link")
doubleclick(ListItem("Directories"))
doubleclick(Point(200, 300))
doubleclick(TextField("Username:").center - (0, 20))
```

Strings passed as parameters are automatically wrapped as Text (page 6) objects (so doubleclick("Helium") is equivalent to doubleclick(Text("Helium"))).

drag(*element*, *to*)

Parameters

- **element** – The element to drag.
- **to** – The element or Point to drag to.

Drags the given GUI element to the given location. For example:

```
drag("File", to="Folder")
```

Both parameters "element" and "to" can be of any type that you can pass to `helium.api.click()` (page 7).

The dragging is performed by hovering the mouse cursor over "element", pressing and holding the left mouse button, moving the mouse cursor over "to", and then releasing the left mouse button again.

drag_file(*file_path*, *to*)

Drags a file from local computer over an HTML element specified by its name and drops it there. This allows, for example, to attach files to emails in Gmail:

click("COMPOSE") write("example@gmail.com", into="To") write("Email subject", into="Subject") drag_file("C:Documentsnotes.txt", to="Drop files here")

find_all(*predicate*)

Lets you find all occurrences of the given GUI element predicate. For instance, the following statement returns a list of all buttons with label "Open":

```
find_all(Button("Open"))
```

In a typical usage scenario, you want to pick out one of the occurrences returned by `find_all()` (page 8). In such cases, `list.sort()` can be very useful. For example, to find the leftmost "Open" button, you can write:

```
buttons = find_all(Button("Open"))
leftmost_button = sorted(buttons, key=lambda button: button.x)[0]
```

get_driver()

Returns currently used Selenium driver object.

go_to(*url*)

Parameters **url** (*str*) – URL to open.

Opens the specified URL in the current web browser window. For instance:

```
go_to("www.google.com")
```

highlight(*element*)

Parameters **element** – The element to highlight.

Highlights the given element on the webpage by drawing a red rectangle around it. For example:

```
highlight(Text("Helium"))
```

hover(**elements*)

Parameters ***elements** – Comma-separated list of GUI elements, labels (strings) or points.

Consecutively hovers the mouse cursor over the given elements. For example:

```
hover("Close")
hover(Button("OK"))
hover("Home", "Download", "Start")
hover("Home", MenuItem("Download"), "Start")
hover(Point(200, 300))
hover(ComboBox("File type").center + (50, 0))
```

Strings passed as parameters are automatically wrapped as `Text` (page 6) objects (so `hover("Helium")` is equivalent to `hover(Text("Helium"))`).

press(**keys*)

Parameters ***keys** – Key or comma separated list of keys to be pressed.

Presses the given keys in sequence. To press a normal letter key such as 'a' simply call *press* for that character:

```
press('a')
```

You can also simulate the pressing of upper case characters that way:

```
press('A')
```

To press a special key such as ENTER look it up in the list below, then call *press* for it:

```
press(ENTER)
```

To press multiple keys at the same time, concatenate them with +. For example, to press CTRL + a, call:

```
press(CTRL + 'a')
```

To press multiple key combinations in sequence, separate them by commas:

```
press(ALT + 'f', 's')
```

List of non-letter keys:

- **Common keys:** `ENTER`, `SPACE`, `TAB`, `ESC`
- **Modifiers:** `CTRL`, `ALT`, `SHIFT`
- **Insertion / Deletion:** `INS`, `BKSP`, `DEL`, `END`
- **Navigation:** `HOME`, `END`, `PGUP`, `PGDN`
- **Arrows:** `LEFT`, `DOWN`, `RIGHT`, `UP`
- **Other keys above the arrow keys:** `BREAK`, `PAUSE`
- **Function keys:** `F1`, `F2`, ..., `F12`
- **Numpad operations:** `ADD`, `SUBTRACT`, `MULTIPLY`, `DIVIDE`,
- **Numpad numbers:** `NUMPAD0`, `NUMPAD1`, ..., `NUMPAD9`
- **Other numpad keys:** `DECIMAL`
- **Left/right variants of other keys:** `LSHIFT`, `LCTRL`
- **Others:** `CLEAR`, `HELP`, `SEPARATOR`

rightclick(*element*, *select=None*, *then_select=None*, **finally_select*)

Parameters

- **element** – The GUI element, label (string) or point to right-click.
- **select** – GUI element, label (string) or point to left-click immediately after performing the right-click.
- **then_select** – GUI element, label (string) or point to left-click after clicking the `select` element.
- ***finally_select** – Comma-separated list of GUI elements, labels (strings) and points to left-click after clicking the `select` and `then_select` elements

Performs a right click on the given element, optionally clicking on the specified sequence of context menu elements afterwards. For example:

```
rightclick("Something", select="Menu Item")
```

Strings passed as parameters are automatically wrapped as `Text` (page 6) objects (so `rightclick("Helium")` is equivalent to `rightclick(Text("Helium"))`).

scroll_down (*steps=1*)
Scrolls down the mouse wheel given number of steps.

scroll_left (*steps=1*)
Scrolls left the mouse wheel given number of steps.

scroll_right (*steps=1*)
Scrolls right the mouse wheel given number of steps.

scroll_up (*steps=1*)
Scrolls up the mouse wheel given number of steps.

select (*combo_box*, *value*)
Selects specified value from a combo box. For example:

```
select("Language", "English")
select(ComboBox("Language"), "English")
```

set_driver (*driver*)
Sets the Selenium driver.

start_chrome (*url=None*)

Parameters **url** (*str*) – URL to open.

Starts the Chrome web browser and opens the specified URL, if provided. For instance:

```
start_chrome("http://www.google.com/")
```

start_firefox (*url=None*, *firefox_profile=None*)

Parameters

- **url** (*str*) – URL to open.
- **firefox_profile** (*selenium.webdriver.firefox.firefox_profile.FirefoxProfile*) – Selenium's Firefox profile.

Starts the Firefox web browser and opens the specified URL, if provided. For instance:

```
start_firefox("http://www.google.com/")
```

start_ie (*url=None*)

Parameters **url** (*str*) – URL to open.

Starts the Internet Explorer web browser and opens the specified URL, if provided. For instance:

```
start_ie("http://www.google.com/")
```

switch_to (*window*)

Parameters **window** – The title (string) of a window on screen or a `Window` (page 7) object

Switches to the given window. For example:

```
switch_to("Google")
```

This searches for a window whose title contains "Google", and activates it.

wait_until (*condition_fn*, *timeout_secs=10*, *interval_secs=0.5*)
Waits until the given condition function evaluates to true. This is most commonly used to wait for a GUI element to exist:

```
wait_until(Text("Finished!").exists)
```

When the optional parameter `timeout_secs` is given and not `None`, `wait_until` raises `TimeoutExpired` if the condition is not satisfied within the given number of seconds. The parameter `interval_secs` specifies the number of seconds Helium waits between evaluating the condition function.

`write`(*text*, *into=None*)

Parameters

- **text** (*one of (str, unicode)*) – The text to be written.
- **into** (*one of (str, unicode)*) – Name or label of a text field to write into.

Types the given text into the active window. If parameter 'into' is given, first sets the focus to the text field with name or label specified by this parameter. Common examples of 'write' are:

```
write("Hello World!")
write("user12345", into="Username:")
```